A Formal Approach using PEPA to Performance Analysis of Service-Oriented Architecture Style Specified through Graph Transformation System

Mr. Bhramarabara Biswal, Mrs. Debarchana Rout

Department of Master in Computer Application, College Of Engineering Bhubaneswar, Odisha, INDIA.

ABSTRACT

Due to the proliferation of widely dispersed, concurrent software systems and the requirement for having software that is deemed efficient, performance evaluation must be conducted from the early stages of software development before implementation. It would be time-consuming and economical to modify the program after it has been implemented in order to boost efficiency. Because it is the most abstractive of the current architectural styles, the service-oriented style is the greatest option for dealing with extensiveness and distributedness. Architectural modeling can be done with a formal, understandable, dynamic language called Graph Transformation System (GTS). In this paper, we have introduced a method that uses the PEPA language to evaluate the service-oriented architectural style's performance using a graph transformation system as a model. In order to evaluate performance through PEPA, systems behavior and structure that have been extracted from the graphs must be identified. This requires converting the architectural model provided by graph transformation systems into the PEPA performance model, a formal modeling language based on process algebra. In conclusion, the action throughput of software systems, as well as the condition of use and capacity utilization of each component, have been studied and corresponding charts have been produced.

Key Words: Graph Transformation System, PEPA, SaaS

1. INTRODUCTION

In conventional methods, performance analysis is carried out after the system has been put into use. If the software system performs poorly, this requires repeating the designing process, which is both time- and money-consuming. The contemporary methodology, known as software performance engineering, forecasts performance analysis during the first phases of development and before going live . Architectural models are analyzed in performance engineering, a procedure that eliminates all details and deals only with the total. The following are some of the attributes that performance engineering frequently looks at:

i**. Throughput:** the quantity of transactions that obtain services within a given time frame.

ii. **Utilization:** the percentage of time a resource wastes providing services to a client in relation to the total amount of time.

Performance evaluations on several architectural styles have shown that the service-oriented style, with its high degree of abstraction, is the greatest option for handling the complexity, extendedness, and distributed nature of today's systems. Although there are some parallels between the ideas of object and component and the concept of service, as Figure 1 illustrates, the service-oriented style has a higher abstraction level than both object-oriented and component-based styles.

PEPA is a formal language appropriate for modeling and performance evaluation that is based on process algebra. Other approaches to performance modeling include petri nets and queuing networks. Nonetheless, the current paper uses process algebra for the following reasons:

1. **Formality:** the mathematical foundation that ensures its accuracy, correctness, and clarity.

2. **Abstractness:** taking out the specifics and using the generals of the system—that is, its components and interactions—to build a performance model.

3. **Compositionality:** building the structure through mechanisms via which groups of subsystems communicate, simplifying the model.

PEPA(PERFORMANCE EVALUATION PROCESS ALGEBRA)

PEPA language is utilized for performance modeling in this paper. Process algebra is the foundation of this language. This language's characteristics make it the most suitable for performance evaluation at the level of architecture. The abstract nature of language is the most significant of these.

The PEPA system is composed of subsystems that interact collectively. In this language, every action has an action type, which is always shortened to type. The time period for each task in this language is a stochastic variable with an exponential distribution. Rate is the abbreviation for this metric, which is also known as activity rate.

2. EXTRACTING STRUCTURAL ELEMENTS

The process for modeling the structural elements and communicative restrictions of the service-oriented style will be covered in this part. Component, connector, port, interface, and operation are the parts of a service-oriented style. Information about the system and its processes are components. Interfaces provide access to component processes that can be presented.

Connectors facilitate communication between components. The component's internal structure is concealed and only accessible through its ports. A connector is used to connect two interfaces together.

EXTRACTING BEHAVIORAL ELEMENTS

Since architectural elements of a service-oriented style are only able to communicate through communicative mechanisms—such as connectors, query sending, query results receiving, and so forth—all communicative mechanisms can be considered to be part of style behavior.

The whole list of communicative mechanisms in the service-oriented style, along with an explanation of each, is as follows:

Opening port: this function is invoked when a component communication is required, designating the proper port for that purpose.

Closing port: the port is released when there is a breakdown in contact.

Communication establishment: establishing a communication channel between two elements. Disconnection of the communication channel between two components is known as communication disconnection.

Call operation: the operation seeking component sends a call request pertaining to a single operation on a single component.

Receive call: the operation supplying component will receive a call request for a single operation on a single component.

Sending a response involves getting ready and sending the right answers for an operation that has been requested from the requestor component to the operation provider component.

3. TRANSFORMING THE PROPOSED MODEL INTO A PERFORMANCE MODEL

Components and operations, or other actions and their combinations inside the system for establishing a general system and its evaluation, are the main components of PEPA language. The two stages of its transforming algorithm are as follows:

Finding and extracting the performance model's structural components.

Send service query: The service requester component sends a service query message to service discovery.

Receive service query: The service requester sends a message with a service query to service discovery.

Send query result: The service requester receives a response message for the query result from service discovery.

Receive query result: The message with the query's outcome is sent to the service requester.

A. Identifying and extracting structural components of performance model

As is well known, the PEPA language is made up of a wide range of activities.

Since the concepts of component type in the performance model are equivalent to these elements in the graph scheme (i.e., component and component type), searching must begin at the beginning in order to find them.

The amount of name property in the graph on page 26 is added to the list of PEPA model components for each component type.

Since the general equation of PEPA requires knowledge of the number of participating components in the system, one can use the number of node components of each component type as the number of corresponding components when examining the start graph for the model.

B. Extracting interactive behavior of components and creating system equation

In this phase, the interactive behavior is first extracted, and then the PEPA language is used to generate the system equation. Considering the facts

Graphs are used to extract the system's interactive activity, which looks like this:

Customer asks travel agency for itinerary

Client = (journeyrequest, r).Journey Agency

The travel company answers the client's inquiry.

Agency for Travel = (tsendresponsetoc, r).

Travel agent receives the booking voyage request from the customer.

(csendbookjourneytot, r) is the customer.

Tour Operator Requests from travel agencies to fly with airlines

Operator of Travel = (requestflight, r).Airline

Airline answers request from travel agency

(asendresponsetot, r) = airline. Journey Agency

Travel agency notifies airline about flight booking

Tour Operator = (bookflight, r). Airline

Travel agent receives the airline's reservation and ticket (Airline = (sendticket, r).

Tour Operator

The customer receives flight documentation from the travel agent.

Transport Organization = (tsenddocumenttoc, r).

Client

Here is the system equation:

((Trip Agency / Airline < requestjourney, tsendresponsetoc, csendbookjourneytot, tsenddocumenttoc>)

4. PERFORMANCE EVALUATION

This section will assess the model we suggested for the electronic travel agency case study. Figure 9 shows the system throughput chart for various actions. The throughput for eight major activities (apart from those necessary for communication) is the same, indicating that the service-oriented style is balanced in its behavior with respect to its actions. The travel agency component that allows for speedy communication has the highest throughput when it comes to opening and closing ports both before and after conversation.

5. CONCLUSIONS

In this study, we presented a method for evaluating the service-oriented architectural style's performance using a graph transformation system modeled by the PEPA language.

Initially, a graph transformation system modeling pattern for service-oriented architectural style has been presented. Subsequently, a technique has been presented that allows the current model to be converted into a performance analysis model, effectively turning the architectural model into a performance model. In this work, the architecture was described using a graph transformation system, and PEPA was employed for performance modeling and the two clear-cut, formal approaches.

In this work, the architecture was described using a graph transformation system, and PEPA was employed for performance modeling and the two clear-cut, formal approaches.

6. REFERENCES

1. T. Kauppi, ”Performance analysis at the software architectural level,” Technical report, ISSN: 14550849, 2003.

2. R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Software Architecture: Foundations, Theory, and Practice, New York, NY, USA: John Wiley, 2008.

3. S. Thöne, “Dynamic Software Architectures A Style-Based Modeling and Refinement Technique with Graph Transformations,” Ph.D thesis, University of Paderborn, Paderborn, Germany, 2005.

4. V. Rafe, “Senario-driven analysis of systems specified through graph transformation,” Visual Language and Computing, vol. 24, pp. 136-145, 2013.

5. V. Rafe, and F. Mahdian, “Style-based modeling and verification of fault tolerance service oriented architectures,” Procedia Computer Science, vol. 3, pp. 972-976, 2011.

6. J. Hillston, “ Tuning Systems :From Composition to Performance,” The Computer Journal, vol. 48, pp. 385-400, 2005.